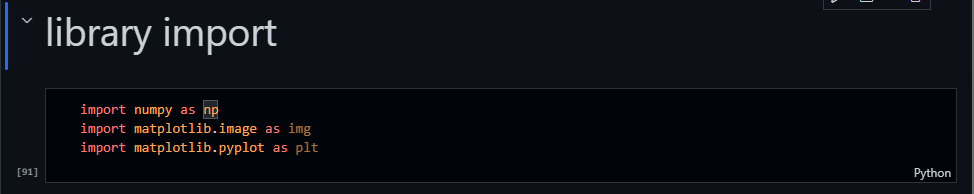
**패턴인식**

**KNN**



|  |
| --- |
| 20173709 나원후 |

KNN프로젝트는 소스코드가 크게 두가지가 있습니다.   
Jupyter notebook을 중심으로 개발하였고, 나중에 실행파일로 변환하기 위해 .py 파일로 포팅하였습니다. 설명은 jupyter notebook 위주로 진행하겠습니다.   
Jupyter notebook에서는 데이터 분석, 가장 적절한 k 값을 찾는 과정, 해당 k값을 사용했을 때의 정확도, 예측값과 정답이 저장된 파일 저장, 그리고 KNN라이브러리 중, sklearn을 이용하여 제 코드에서 나온 데이터와 비교합니다.   
main.py로 만든 main.exe파일은 위의 내용은 수행하지 않고, 가장 적절한 k에 대해서 산출한 결과를 반환합니다.

1. 라이브러리 import  
개발에 필요한 라이브러리를 import합니다. 빠른 연산을 위한 넘파이와, 결과를 출력하기 위한 matplotlib을 사용하였습니다.

2. Data loading  
텍스트, 스크린샷, 모니터, 화면이(가) 표시된 사진

자동 생성된 설명  
satisfaction\_data.csv 로부터 데이터를 로드합니다. 이때 각 column의 이름과 원본 데이터는 공개 되어있지 않기 때문에, 이름은 지정하지 않습니다. class정보를 담고있는 7번째 열은 문자열로 저장이 되어있습니다. 한꺼번에 로드하면 데이터타입 오류가 나므로 따로 로드해줍니다.

텍스트, 모니터, 스크린샷, 화면이(가) 표시된 사진

자동 생성된 설명3. Data instpection   
데이터 분석을 진행합니다. 이 셀은 각 column에 대해서 최소, 최대, 평균, 분산을 출력해줍니다. 그리고 normalization을 위한 minmax를 저장합니다. 다음은 inspection의 결과입니다.

텍스트이(가) 표시된 사진

자동 생성된 설명 텍스트이(가) 표시된 사진

자동 생성된 설명

우선, 유클리드 거리로 데이터간 거리를 구한다면 col\_3에 있는 데이터가 상대적으로 다른 데이터보다 100~10000배가량 크기 때문에, col\_3의 데이터 간 거리가 다른 feature간 거리를 trivial하게 만듭니다. Feature에 있는 값들이 모두 동일한 영향력을 행사하도록 normalization을 진행합니다. 즉, 최대가 1, 최소가 0이 되도록 값을 모두 조정합니다.

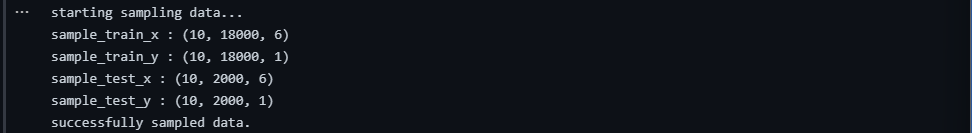
텍스트이(가) 표시된 사진

자동 생성된 설명4. Normalization  
이전에 data inspection에서 구한 각 column의 min, max로 normalization을 수행합니다.

텍스트, 스크린샷, 화면, 검은색이(가) 표시된 사진

자동 생성된 설명5. Augmentation  
저는 train, test data set을 나누기 위해서 다음과 같이 수행하였습니다. 먼저, feature와 class를 이어 붙인 행렬을 만든 후, 그 행렬을 무작위로 섞습니다.

6. Dataset separation  
텍스트이(가) 표시된 사진

자동 생성된 설명  
그리고 섞인 결과물을 1:9로 나누어 각각 sample\_train\_x, sample\_train\_y, sample\_test\_x, sample\_ test\_y에 저장합니다. 이 과정을 10번 수행하여 총 10개의 샘플을 가지도록 합니다.  


7. Compute distance by matrix  
텍스트이(가) 표시된 사진

자동 생성된 설명  
각 sample\_test\_x에 대해서 모든 sample\_train\_x에 대한 거리를 구합니다. 저희는 총 10개의 샘플 묶음이 있으므로 10번 반복합니다. 여기서 두 샘플사이의 거리를 구하기 위해 행렬을 사용하였습니다. 원하는 값은 10개의 샘플에 대해서 2000개의 test case마다 train\_x사이의 거리를 저장한 행렬입니다. 행렬의 요소 하나는 다음과 같은 형식을 지닙니다.

(a-a’)^2 + … + (f -f’)^2 = (a^2 + … + f^2) -2\*(a\*a’ + … + f\*f’) + (a’^2 + … + f’^2)

따라서 총 3가지 요소로 나눌 수 있습니다. 각각 sample\_test\_x요소의 제곱들의 합, sample\_test\_x와 sample\_train\_x요소들의 곱, sample\_train\_x요소의 제곱들의 합을 의미합니다.

넘파이의 broadcasting을 이용해서 3중for문을 이용했을 때보다, 1개의 for루프만을 사용하여 1시간에서 5.7초로 크게 시간을 줄일 수 있었습니다.

계산된 각 거리는 euclid\_dists\_matrix에 저장합니다.

이제 원소별로 거리를 구했으니, 가장 가까운 위치의 인덱스들을 찾습니다. 그리고, 그 인덱스를 sample\_train\_ y에 넣어 class를 구합니다. 구한 클래스 중 가장 많이 나온 클래스를 예측값으로 지정합니다.

이때, k의 개수가 짝수이면 분류가 2가지이기 때문에 1:1로 나올 수 있습니다. 따라서 k의 값은 홀수로 설정합니다.

텍스트이(가) 표시된 사진

자동 생성된 설명8. compute prediction  
지정한 k에 대해서 예측을 수행합니다. 여기서 함수를 두가지를 사용합니다.

np.argsort는 입력으로 받은 행렬을 오름차순으로 정렬하여, 정렬되기 전의 인덱스를 반환합니다. euclid\_dists\_matrix의 i row (i 번째 샘플), j col (j번째 test\_x) 에는 j 번째 sample\_test\_x에 대해서 모든 sample\_train\_x까지의 거리를 저장하고 있습니다. 즉, 18000개의 데이터를 갖고있습니다.  
이 18000개의 데이터를 오름차순으로 정렬하여 정렬된 요소들의 정렬되기전의 인덱스를 알 수 있다면, 그 인덱스를 참조하여 sample\_train\_y에 넣어 class를 추정할 수 있습니다. 여기서 참조할 인덱스의 개수가 바로 k 가 됩니다. 참조할 만큼 column을 가져와 nearest\_class 에 저장합니다.

np.bincount는 양의 정수로 구성된 넘파이 배열에서 각각의 빈도수를 count하여 그 배열을 반환합니다. 결국 반환값은 [n, m]의 형태가 됩니다. 여기서 n은 0인 값들의 개수, m은 1인 값들 의 개수입니다. 각각은 클래스의 개수를 의미합니다.  
9. Compare prediction and real class  
텍스트이(가) 표시된 사진

자동 생성된 설명  
이제 정답과 추정값을 비교하여 정확도를 측정합니다.

sample\_test\_y에는 정답이 들어있습니다. 그 값에서 prediction값을 뺐을 때, 요소의 값이 0이라면 정답입니다. 따라서 0인 값들만 count한 후, 전체 test의 개수만큼 나누어 정답률을 산출합니다.

10. 가장 적합한 k찾기  
프로그래밍할 때 k의 배열을 입력받아 k 마다 iteration을 돌 수 있도록 설계하였습니다. K를 1부터 시작하여 홀수로 1500가량까지 확인해 본 결과, 약 111 까지는 정확도가 최대 80까지 오르다가 이후로는 overfitting되어 점점 정확도가 떨어지는 모습을 보였습니다. 따라서 저는 main.exe로 포팅할 때 k 의 값을 101로 고정하여 KNN을 수행하도록 설계하였습니다. 아래는 수행한 결과들입니다.

11. compute by hamming distance  
만약 거리를 distance metric을 euclid distance로 계산하지 않고, hamming distance로 계산했을 때 정확도를 추정합니다.